Towards Active Participation of Domain Experts in Modeling Language Evolution

Malvina Latifaj
malvina.latifaj@mdu.se
Mälardalen University
Västerås, Sweden

Abstract

The evolution of domain-specific modeling languages is crucial for maintaining their relevance and effectiveness as systems and requirements evolve. Typically, this process requires collaboration between domain experts and developers, but it often faces misunderstandings, misaligned goals, and communication barriers. These challenges can lead to language evolution misaligned with domain experts' intentions, stemming from their limited participation due to a lack of technical skills. Participatory modeling practices can mitigate these issues by involving domain experts throughout the entire language evolution process. We propose advancing to the self-mobilization level of participatory modeling, enabling domain experts to independently drive language evolution. This advancement requires tools that eliminate the need for technical metamodeling expertise. In this vision paper, we describe MetaMorph, a tool aimed to support domain experts in achieving self-mobilization in the evolution of domain-specific modeling languages. MetaMorph allows domain experts to specify changes to the underlying metamodel through existing model instances, providing instant feedback on the resulting evolved model. It also supports the automatic evolution and co-evolution of metamodels and models. MetaMorph aims to minimize the risk of inappropriate language evolution, accelerate the overall process, and enhance the adoption and acceptance of evolved languages.

CCS Concepts

• **Software and its engineering** → *Domain specific languages.*

Keywords

Model Driven Engineering, Participatory Modeling, Collaborative Modeling, Domain Specific Modeling Languages, Metamodel Evolution, Model Co-evolution

ACM Reference Format:
Malvina Latifaj. 2024. Towards Active Participation of Domain Experts in Modeling Language Evolution. In *ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24), September 22–27, 2024, Linz, Austria.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3652620.3688563

MODELS Companion '24, September 22–27, 2024, Linz, Austria

ACM ISBN 979-8-4007-0622-6/24/09
https://doi.org/10.1145/3652620.3688563

1 Introduction

Model-Driven Engineering (MDE) [21] has established itself as a powerful paradigm for the development of complex systems, leveraging Domain-Specific Modeling Languages (DSMLs) to enhance productivity, improve stakeholder communication, and ensure alignment with domain-specific requirements. DSMLs are designed to encapsulate domain concepts and rules, thereby providing high-level abstractions that simplify modeling tasks [10]. As systems evolve and new requirements emerge, DSMLs must undergo evolution to remain relevant and effective. This evolutionary process is essential for maintaining their utility and efficacy in representing the current state of the domain [7]. DSMLs, or modeling languages, can evolve in terms of abstract syntax, concrete syntax, semantics, and tool support. We focus on the abstract syntax, which defines the concepts and relationships within the language.

The evolution of DSMLs requires a deep understanding of both domain-specific knowledge and the technical aspects of language engineering [18]. In the absence of an individual skilled in both areas, this process necessitates collaboration between domain experts, who possess extensive domain knowledge, and developers, who have the technical expertise needed to implement changes. However, as in any scenario where one group of people (i.e., developers) creates something that another group of people (i.e., domain experts) relies on, significant challenges can arise [24]. Due to their lack of technical skills in language engineering, domain experts' participation is often limited to the initial decision-making and analysis stages, and the final deployment phase, where they test the language [8]. While collaboration between domain experts and developers exists, communication barriers, misunderstandings, and misalignments can lead to inappropriate language evolution that does not align with the domain experts' intentions. The exclusion of domain experts from the design and implementation phases exacerbates this issue, resulting in iterative cycles and potential delays as both parties strive to reach a mutual understanding and ensure proper outcome. These challenges can impede the efficient and accurate evolution of DSMLs.

Participatory Modeling (PM) is defined as *"a purposeful learning process for action that engages the implicit and explicit knowledge of stakeholders to create formalized and shared representations of reality"* [25]. While PM is typically employed to address complex social and environmental problems [5, 6, 17], its application in the context of modeling language evolution can ensure that languages remain relevant and effective by incorporating the latest domain knowledge through active participation of domain experts. Mapping the involvement of domain experts in the construction and evolution of modeling languages to Pretty's participation typology [19], it has, in the majority of cases, reached the levels of functional and

interactive participation at best [8, 23]. In this context, domain experts are actively engaged in discussions and decision-making; however, unless they possess the necessary technical skills, their involvement often stops short of the technical implementation of changes, thereby limiting their impact on the final outcomes.

To fully realize the benefits of PM in language evolution, it is essential to advance to the level of self-mobilization in Pretty's topology. In this context, self-mobilization refers to empowering domain experts to drive the evolution of modeling languages without relying on technical developers. At this level, domain experts would not only provide feedback and participate in discussions but also take an active role in the technical implementation and continuous improvement of the language. Developers, on the other hand, could focus on more pressing technical matters, or provide support when needed. Achieving this level of autonomy for domain experts necessitates tools that eliminate the need for deep technical expertise. While example-driven metamodel development approaches aim to include domain experts in the design process by inducing metamodels from examples they create [9, 15, 20], these methods are more suitable for building the modeling language from scratch. They are less efficient for evolution scenarios where real-world models already exist, since creating example fragments for refinement can be time-consuming, effort-intensive, and may not accurately represent real-world scenarios. By experience, domain experts prefer to express language evolution by demonstrating desired changes directly in their existing models.

In this paper, we describe MetaMorph, a tool envisioned to facilitate reaching self-mobilization levels for domain experts with respect to language evolution. MetaMorph empowers domain experts to specify changes required for modeling language evolution using practical model instances, rather than relying on examples, and provides instant feedback on the resulting evolved model. Furthermore, MetaMorph automates the generation of the evolved language and the corresponding model co-evolution mechanisms, thus streamlining the process and minimizing potential errors. By providing domain experts with the capability to directly influence both the conceptual and technical aspects of DSML's evolution, MetaMorph leads to more accurate representations of the domain, shorter development times, faster prototyping, and a continuous alignment of the DSML with evolving domain requirements.

The remainder of this paper is structured as follows. Section 2 describes MetaMorph's requirements. Section 3 presents the MetaMorph approach. Section 4 investigates the related work and Section 5 concludes the paper and describes future directions.

2 MetaMorph Requirements

The following initial set of requirements outlines the features that MetaMorph must possess to fulfill its intended purpose.

RQ_1. Support the specification of changes through model instances.
MetaMorph must allow domain experts to specify changes to the underlying abstract syntax (i.e., metamodel) by modifying model instances they use in practice. By directly altering familiar model instances, domain experts can more easily and effectively convey their desired evolution of the modeling language.

RQ_2. Implement inconsistency management strategies.
MetaMorph must incorporate inconsistency management strategies to handle conflicts from domain experts' changes to model instances that can create ambiguity in translating the changes to the underlying metamodel. Effective inconsistency management will prevent such conflicts, ensuring accurate metamodel evolution.

RQ_3. Support the generation of the evolved metamodel.
MetaMorph must translate changes made to the model instance into corresponding changes in the initial metamodel and automatically generate an evolved metamodel that incorporates the changes. This automates tasks traditionally handled by developers, allowing domain experts to directly influence and evolve the metamodel.

RQ_4. Support the generation of model co-evolution mechanisms.
MetaMorph must support the automatic generation of model co-evolution mechanisms between the initial and evolved metamodel. These mechanisms should enable seamless translation of model instances to the evolved metamodel, ensuring conformity to the updated structure. This capability is crucial for maintaining the integrity and usability of models throughout the evolution process.

3 MetaMorph Approach

MetaMorph comprises four components, each with distinct responsibilities: *MetaLab*, *MetaFix*, *MetaBuild*, and *MetaSync*. Figure 1 illustrates the overall workflow, and the subsequent sections offer detailed descriptions of each component.

3.1 MetaLab

MetaLab is an environment designed to facilitate the specification of changes to the underlying metamodel through model instances used in practice, thereby addressing RQ_1, and serves as the user interface for MetaMorph. Domain experts, hereafter referred to as users, given their role as the primary operators of this tool, begin by importing the metamodel to be evolved ($DSML_1$) along with a conforming model instance (M_{DSML_1}), which they shall use to specify the necessary changes.

MetaLab features a dual side-by-side view interface, with both views initially envisioned to display information in a tree-based editor. The *base* view displays the original model instance (M_{DSML1}), which conforms to the imported metamodel ($DSML_1$). The *mock* view presents an exact copy of the original model but differs in two key aspects: (i) it enables the specification of required changes, and (ii) the mock model[1] ($MDSML_2$) can deviate from conformance with the imported metamodel ($DSML_1$), offering the flexibility needed to describe changes. This dual-view interface allows users to visualize the evolution of the mock model in comparison to the original.

The mock view in MetaLab is tailored for domain experts with no metamodeling expertise. Users will have access to various options for modifying each model element, tailored to the type of change being made. MetaLab supports a wide range of modifications, including renaming concepts, changing types, adding or removing child elements, merging or splitting elements, making elements abstract, and creating, updating, or deleting relationships between

[1]Mock models refer to models used by domain experts to specify the changes to the underlying metamodel.

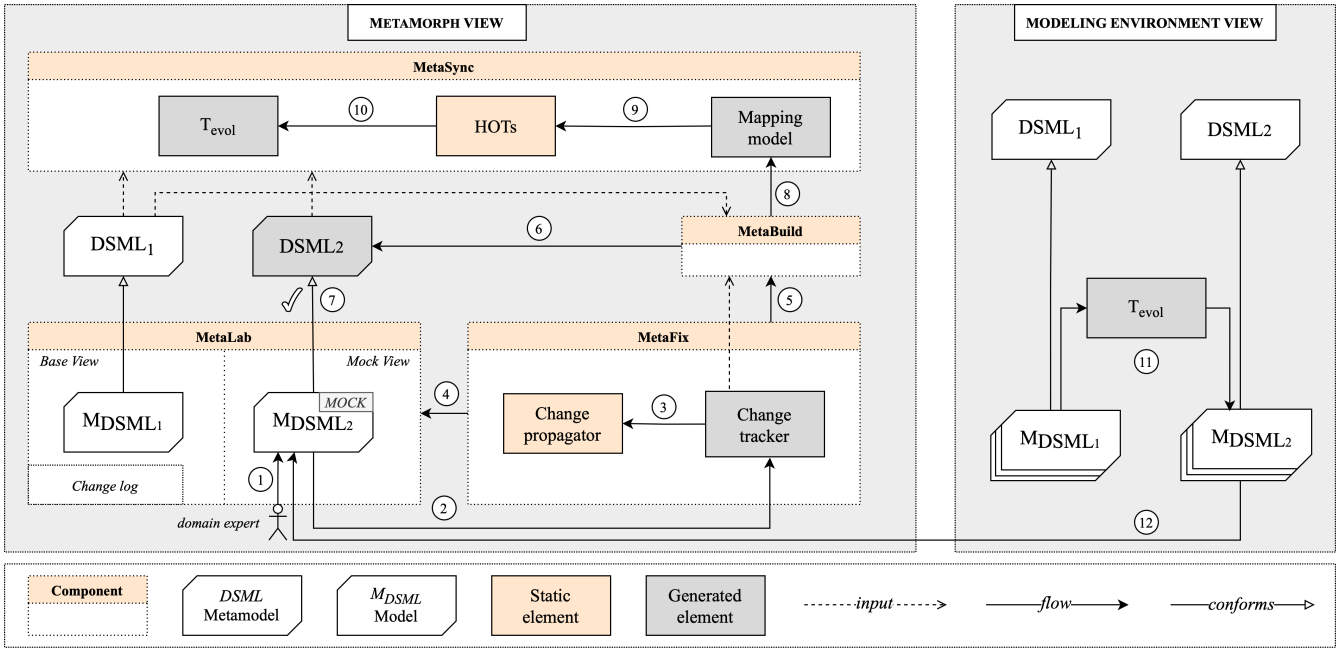

Figure 1: MetaMorph Workflow

elements. Depending on the nature of the change, additional parameters and settings will be presented to collect the necessary information. Although these modifications are made at the model level, they are guided by existing literature on metamodel evolution [2, 3, 26], as they are ultimately translated into metamodel changes. The mock view in MetaLab is meticulously designed to ensure that information is complete and accurate while being presented in a user-friendly manner tailored to the expertise of domain experts. This design conceals the complexities of metamodeling, focusing instead on the models themselves, which domain experts understand well. By requesting and allowing users to provide information in straightforward, simplified ways, MetaLab centers the entire process around the models, making it intuitive and accessible for domain experts. Furthermore, the mock view offers a flexible editing environment that does not enforce conformance checks, allowing users to freely modify elements, relationships, and properties. This flexibility is crucial for enabling domain experts to intuitively specify necessary modifications without being constrained by the imported metamodel.

Additionally, MetaLab includes a change log view that chronologically lists all modifications made to the mock model. This view enables users to track the evolution of their changes over time, with each entry providing the option to revert the model to its state immediately before that entry. This capability allows users to experiment confidently, knowing they can always revert to a previous version if necessary, and enhances the overall user experience by providing clarity, control, and flexibility throughout the language evolution process.

3.2 MetaFix

MetaFix is the component responsible for tracking and propagating all changes made by users in the mock model. Once a user modifies the model instance within MetaLab to reflect desired metamodel evolutions, these changes are systematically captured and recorded by MetaFix's *change tracker* (step ②). This tracker maintains a detailed change log that documents the ongoing evolution of the metamodel. The recorded changes are subsequently fed into MetaFix's *change propagator* (step ③), which processes this information and applies the modifications to every instance of the affected concept within the mock model (step ④). The latter process is essential for accurately reflecting the changes specified in one instance to all the affected instances in the model, ensuring transparency for the user and minimizing the risk of inconsistencies. For example, consider a customer relationship management (CRM) modeling language where a user decides to rename the concept *Customer* to *Client*. If this change is not propagated to all instances within the model, the user might inadvertently rename *Customer* to *Buyer* in another instance. Such conflicting changes create ambiguity and inconsistencies in how these changes should be translated in the underlying metamodel. MetaFix addresses this issue by ensuring that once a concept is modified, the change is consistently applied across all instances, thus preventing conflicts and directly addressing RQ_2.

Given the potentially large size of the model instances used in practice by domain experts, propagating changes after each modification could be computationally heavy. Therefore, in MetaFix, we consider various propagation strategies: changes may be propagated incrementally, deferred to a later point, initiated manually by the domain expert, or limited to a subset of the model designated as the active mock model, which can be defined by the user in MetaLab. While other inconsistency management strategies could

have been chosen – such as designating a specific instance as a pivot and making changes to the underlying concept only through that instance, or notifying the user if they attempt to change a concept that has already been modified – the current approach has been selected to provide transparency to the user. By showing how changes affect the entire model instance, this method gives users better insights into whether the changes achieve the desired outcome, thus avoiding confusion and potential frustration.

3.3 MetaBuild

MetaBuild is the component responsible for the automatic generation of the evolved metamodel, thereby addressing RQ_3. Once the domain expert is satisfied with the modifications made to the mock model, the generation of the evolved metamodel can be initiated (step ⑤). MetaBuild plays a critical role as it translates user-made modifications at the model level (step ①), recorded in the change tracker, into corresponding changes in the metamodel ($DSML_1$), thereby inferring a new evolved metamodel ($DSML_2$). This translation is achieved through a well-defined mapping mechanism within MetaBuild, which ensures that changes at the model level are consistently and accurately reflected at the metamodel level. This mapping is essential to prevent ambiguity and ensure deterministic outcomes in the evolution process. Consequently, the accurate and comprehensive collection of information in MetaLab is paramount, as it directly impacts the clarity and effectiveness of the transformations performed by MetaBuild.

If the generated metamodel is valid and the mock model (M_{DSML_2}) conforms to it (step ⑦), MetaBuild can proceed to step ⑧ where it initiates the generation of the synchronization infrastructure between the original and evolved metamodels, as detailed in Section 3.4. Should any logged changes result in an invalid metamodel (e.g., two concepts with the same name), MetaBuild produces an error log that pinpoints the specific change and its associated issue, enabling the domain expert to identify and rectify problematic changes using MetaLab. This iterative process of correction is essential for achieving a valid metamodel that accurately reflects the desired evolutions.

3.4 MetaSync

MetaSync is the component responsible for the automatic generation of model co-evolution mechanisms, thereby addressing RQ_4. Once the evolved metamodel is successfully inferred, MetaBuild proceeds to generate a mapping model that contains the correspondences between the original ($DSML_1$) and evolved ($DSML_2$) metamodel (step ⑧). The mapping model serves as an input to higher-order transformations (HOTs) [22] (step ⑨). HOTs are sophisticated model transformations that generate other transformations, leveraging the meta-information provided by the mapping model. Specifically, these HOTs analyze both the original and evolved metamodels and process the mapping model to generate the required model transformation (T_{evol}) in step ⑩. This infrastructure is grounded in our prior work [13], where we proposed a dedicated mapping modeling language which provides a precise syntax and semantics for specifying correspondences between metamodels, ensuring that the mappings are both expressive and unambiguous. Additionally, the HOTs we developed translate this mapping information into an executable model transformation. The co-evolution mechanisms provided by MetaSync will address all three categories of metamodel changes: non-breaking, breaking and resolvable, and breaking and unresolvable [3]. While a survey of existing literature lists various approaches to managing breaking and unresolvable changes [4], in MetaMorph, we explore an innovative approach of deriving resolution techniques directly from user-defined changes in the mock model, aiming to achieve higher levels of PM in the model co-evolution process as well.

After the domain expert has evolved the modeling language as expected and is satisfied with the results, the model transformation (T_{evol}) generated by MetaSync is used in step ⑪ to transform models conforming to the original metamodel ($DSML_1$) to models conforming to the evolved metamodel ($DSML_2$). This ensures that models co-evolve with the metamodel. If the evolved models do not reflect the domain experts' expectations, and this issue was not identified in the mock model, the domain experts can return to MetaLab in step ⑫ to specify the necessary changes.

4 Related Work

Izquierdo and Cabot [8] introduce a collaborative platform for the development of DSMLs. This platform utilizes Collaboro, a DSL designed to capture change proposals, potential solutions, and feedback throughout the development phases. While this method aims to enhance communication between technical developers and domain experts, the implementation of changes within the metamodel still relies on technical developers. In contrast, our approach empowers domain experts to specify changes directly on model instances, which are then translated into modifications in the underlying metamodel through automated mechanisms. This removes the dependency on developers and facilitates faster prototyping. Furthermore, Collaboro lacks mechanisms for model co-evolution in response to changes in the metamodel. Lopez et al. [15] and Cuadrado et al. [20] present a methodology for metamodel construction that allows domain experts to specify example model fragments using informal drawing tools such as Dia or yED. These fragments, annotated with hints, clarify the intentions or requirements for specific elements, from which a metamodel is automatically induced and iteratively refined. This approach is also combined with Collaboro by Izquierdo et al. [9]. Although the employed iterative process facilitates the inclusion of new model fragments to evolve the metamodel, it requires that domain experts construct model fragments from scratch. This approach is well-suited for constructing a DSML from the ground up but may be less efficient for evolving an existing DSML with pre-existing model instances, as it requires duplicating these instances or creating new examples using the specified tools. In contrast, our approach allows domain experts to work directly with existing model instances actively used within their domain, eliminating the need to build new examples. This enables them to make informed modifications based on immediate needs. By leveraging existing models, our method ensures that modifications are both relevant and immediately applicable, thus facilitating a more efficient evolution of the modeling language in response to practical demands. The same applies to other related works [1, 16, 27] that induce metamodels or develop complete graphical modeling environments from graphical examples in free-form sketching tools,

as these approaches are typically geared towards the creation of DSMLs from scratch rather than their evolution.

5 Conclusions and Future Directions

In this vision paper, we described MetaMorph, a tool aimed at empowering domain experts in driving metamodel evolution and model co-evolution. This approach seeks to increase their participation in tasks traditionally reserved for developers with specialized metamodeling expertise. By doing so, MetaMorph aims to mitigate the risks associated with improper language evolution, improve language adoption and acceptance, and facilitate rapid prototyping and testing of evolved languages.

The development of MetaMorph presents several significant challenges we plan to address. Foremost is ensuring that the MetaLab environment aligns with the cognitive framework of domain experts while capturing all essential information for metamodel evolution/model co-evolution. Metamodel evolution requires the translation of changes specified at the model level into changes at the metamodel level to infer the evolved metamodel. Model co-evolution requires robust evolution mechanisms and the extraction of resolution techniques to address breaking and unresolvable changes. Additionally, managing large and complex models is crucial to ensure MetaMorph's applicability in practical settings. We are currently focused on deriving a set of changes that can be made at the model level and on specifying how to implement these changes in a manner that aligns with the domain experts' knowledge, emphasizing usability and user experience. Additionally, we are developing a conceptual mapping of these changes to corresponding changes at the metamodel level.

The validation of MetaMorph will proceed through iterative testing phases. Usability testing will involve domain experts interacting with MetaLab to ensure the interface is intuitive and meets their cognitive needs. The propagation mechanisms of MetaFix and the translation mechanisms of MetaBuild to the evolved metamodel and mapping model will be validated to ensure changes are applied accurately and consistently. Performance tests on large and complex models will optimize system efficiency and scalability. Further validation will use industrial use cases. One use case involves our previous work, where we manually evolved the UML-RT metamodel [14] and developed co-evolution mechanisms [11, 12] using models provided by our industrial partner to reflect desired changes. Here, in MetaMorph we will construct the mock model based on these models and automatically generate the evolved UML-RT metamodel and co-evolution mechanisms. This will demonstrate MetaMorph's capability to streamline and automate the evolution of complex modeling languages, ensuring accuracy and efficiency, and enabling domain experts to achieve these outcomes without extensive technical expertise.

References

[1] Hyun Cho, Jeff Gray, and Eugene Syriani. 2012. Creating visual domain-specific modeling languages from end-user demonstration. In *2012 4th International Workshop on Modeling in Software Engineering (MISE)*. IEEE, 22–28.

[2] Antonio Cicchetti, Davide Di Ruscio, Romina Eramo, and Alfonso Pierantonio. 2008. Meta-model differences for supporting model co-evolution. In *Proceedings of the 2nd Workshop on Model-Driven Software Evolution-MODSE*, Vol. 1.

[3] Boris Gruschko, Dimitrios Kolovos, and Richard Paige. 2007. Towards synchronizing models with evolving metamodels. In *Proceedings of the International Workshop on Model-Driven Software Evolution*. Amsterdam, Netherlands, 3.

[4] Regina Hebig, Djamel Eddine Khelladi, and Reda Bendraou. 2016. Approaches to co-evolution of metamodels and models: A survey. *IEEE Transactions on Software Engineering* 43, 5 (2016), 396–414.

[5] Beatrice Hedelin, Steven Gray, S Woehlke, Todd K BenDor, Alison Singer, Rebecca Jordan, Moira Zellner, P Giabbanelli, Pierre Glynn, Karen Jenni, et al. 2021. What's left before participatory modeling can fully support real-world environmental planning processes: A case study review. *Environmental Modelling & Software* 143 (2021), 105073.

[6] Sarah Henly-Shepard, Steven A Gray, and Linda J Cox. 2015. The use of participatory modeling to promote social learning and facilitate community disaster planning. *Environmental Science & Policy* 45 (2015), 109–122.

[7] Ludovico Iovino, Alfonso Pierantonio, and Ivano Malavolta. 2012. On the Impact Significance of Metamodel Evolution in MDE. *J. Object Technol.* 11, 3 (2012), 3–1.

[8] Javier Luis Cánovas Izquierdo and Jordi Cabot. 2012. Community-driven language development. In *2012 4th International Workshop on Modeling in Software Engineering (MISE)*. IEEE, 29–35.

[9] Javier Luis Cánovas Izquierdo, Jordi Cabot, Jesús J López-Fernández, Jesús Sánchez Cuadrado, Esther Guerra, and Juan De Lara. 2013. Engaging end-users in the collaborative development of domain-specific modelling languages. In *Cooperative Design, Visualization, and Engineering: 10th International Conference, CDVE 2013, Alcudia, Mallorca, Spain, September 22-25, 2013. Proceedings 10*. Springer, 101–110.

[10] Steven Kelly and Juha-Pekka Tolvanen. 2008. *Domain-specific modeling: enabling full code generation*. John Wiley & Sons.

[11] Malvina Latifaj, Federico Ciccozzi, Muhammad Waseem Anwar, and Mattias Mohlin. 2021. Blended graphical and textual modelling of UML-RT state-machines: An industrial experience. In *European Conference on Software Architecture*. Springer, 22–44.

[12] Malvina Latifaj, Federico Ciccozzi, Antonio Cicchetti, and Mattias Mohlin. 2023. Cross-Platform Migration of Software Architectural UML-RT Models. In *17th European Conference on Software Architecture ECSA*.

[13] Malvina Latifaj, Federico Ciccozzi, and Mattias Mohlin. 2023. Higher-order transformations for the generation of synchronization infrastructures in blended modeling. *Frontiers in Computer Science* 4 (2023), 1008062.

[14] Malvina Latifaj, Federico Ciccozzi, Mattias Mohlin, and Ernesto Posse. 2021. Towards Automated Support for Blended Modelling of UML-RT Embedded Software Architectures.. In *ECSA (Companion)*.

[15] Jesús J López-Fernández, Jesús Sánchez Cuadrado, Esther Guerra, and Juan De Lara. 2015. Example-driven meta-model development. *Software & Systems Modeling* 14 (2015), 1323–1347.

[16] Jesús J López-Fernández, Antonio Garmendia, Esther Guerra, and Juan de Lara. 2019. An example is worth a thousand words: Creating graphical modelling environments by example. *Software & Systems Modeling* 18 (2019), 961–993.

[17] Guillermo A Mendoza and Ravi Prabhu. 2006. Participatory modeling and analysis for sustainable forest management: Overview of soft system dynamics models and applications. *Forest Policy and Economics* 9, 2 (2006), 179–196.

[18] Marjan Mernik, Jan Heering, and Anthony M Sloane. 2005. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)* 37, 4 (2005), 316–344.

[19] Jules Pretty. 1995. The many interpretations of participation. *Focus* 16, 4 (1995).

[20] Jesús Sánchez-Cuadrado, Juan De Lara, and Esther Guerra. 2012. Bottom-up meta-modelling: An interactive approach. In *Model Driven Engineering Languages and Systems: 15th International Conference, MODELS 2012, Innsbruck, Austria, September 30–October 5, 2012. Proceedings 15*. Springer, 3–19.

[21] Douglas C Schmidt et al. 2006. Model-driven engineering. *Computer-IEEE Computer Society-* 39, 2 (2006), 25.

[22] Massimo Tisi, Frédéric Jouault, Piero Fraternali, Stefano Ceri, and Jean Bézivin. 2009. On the use of higher-order model transformations. In *Model Driven Architecture-Foundations and Applications: 5th European Conference, ECMDA-FA 2009, Enschede, The Netherlands, June 23-26, 2009*. Springer, 18–33.

[23] Maria Jose Villanueva, Francisco Valverde, and Oscar Pastor. 2014. Involving end-users in the design of a domain-specific language for the genetic domain. In *Information System Development: Improving Enterprise Communication*. Springer, 99–110.

[24] Markus Voelter, Sebastian Benz, Christian Dietrich, Birgit Engelmann, Mats Helander, Lennart CL Kats, Eelco Visser, and GH Wachsmuth. 2013. DSL engineering-designing, implementing and using domain-specific languages. (2013).

[25] Alexey Voinov, Karen Jenni, Steven Gray, Nagesh Kolagani, Pierre D Glynn, Pierre Bommel, Christina Prell, Moira Zellner, Michael Paolisso, Rebecca Jordan, et al. 2018. Tools and methods in participatory modeling: Selecting the right tool for the job. *Environmental Modelling & Software* 109 (2018), 232–255.

[26] Guido Wachsmuth. 2007. Metamodel adaptation and model co-adaptation. In *European conference on object-oriented programming*. Springer, 600–624.

[27] Dustin Wüest, Norbert Seyff, and Martin Glinz. 2013. Semi-automatic generation of metamodels from model sketches. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 664–669.